AMERICAN JOURNAL OF AI CYBER COMPUTING MANAGEMENT
Peer Reviewed, Rferred & Indexed Journal
E-ISSN:3069-0102
VOL.6, NO. 2(2026)

Hybrid Ensemble White Shark Optimizer for Fine-Grained Software Bug Detection

N. Rajesh[1], K. Srinivas[2*], Nagapuri Akhil[3], Thota Vamshikrishna[3], Lingampalli Ankitha[3], Varanganti Navya[3]

[1]Assistant Professor, [2]Associate Professor & Head, [3]UG Student, [1,2,3]Department of Computer Science and Engineering (Data science)

[1,2,3]Vaagdevi Engineering College, Bollikunta, Warangal, 506005, Telangana, India

*Correspondence: K. Srinivas (srinivas_k@vecw.edu.in)

ABSTRACT

The increasing complexity of modern software systems has led to a rapid rise in software defects, making reliable bug detection and software classification essential for improving software quality and maintainability. Traditionally, software defect prediction relied on manual inspection, rule-based analysis, and conventional statistical methods. These approaches often required expert intervention, were time-consuming, and struggled to handle large-scale and high-dimensional software metrics data effectively. Machine learning techniques such as Bernoulli Naive Bayes (BNB), Multinomial Naive Bayes (MNB), and Linear Discriminant Analysis (LDA) were later introduced to automate the prediction process; however, these models exhibited limitations in capturing complex nonlinear relationships among software metrics and often produced suboptimal performance on diverse datasets. To address these challenges, this work proposes a hybrid intelligent system, Hybrid Ensemble White Shark Optimizer Convolutional Neural Network Extra Trees (HE-WSO-CNN-ET), designed for efficient bug detection and software classification. The proposed system integrates Convolutional Neural Network (CNN) for feature representation learning, White Shark Optimizer (WSO) for adaptive control, and Extra Trees (ET) for classification. The system processes software metrics such as Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM), Afferent Couplings (CA), Efferent Couplings (CE), Number of Public Methods (NPM), Lines of Code (LOC), and complexity-related attributes to identify defect patterns and classify software types. The proposed approach aims to enhance prediction accuracy, reduce manual effort, and improve scalability in handling real-world software datasets. By leveraging ensemble learning with intelligent feature mapping, the system provides a more robust and efficient solution compared to traditional methods. The experimental results demonstrate improved performance in terms of accuracy, precision, recall, and F1-score, highlighting the effectiveness of the proposed system in software defect prediction and classification tasks.

Keywords: Software Defect Prediction, White Shark Optimizer, Machine Learning, Deep Learning, Convolutional Neural Network.

1. INTRODUCTION

A large-scale global technology outage that occurred in July 2024 was caused by a defect in a software update, leading to significant economic losses and widespread disruption of services across essential sectors such as health care, aviation, finance, and education. This incident clearly demonstrated the critical importance of maintaining high reliability and maintainability in software systems, which form the backbone of modern technological infrastructure. It is estimated that nearly 70% of total software development costs are dedicated to maintenance activities, making it one of the most resource-intensive phases of the software lifecycle. As software systems continue to evolve in size and complexity,

maintaining them has become increasingly challenging due to the substantial growth in codebase and system dependencies. Despite these increasing challenges, software maintainability remains a crucial factor in determining the success and longevity of software products. One of the key factors influencing maintainability is the presence of defects. Software defects significantly impact system stability and increase the effort required for maintenance. By reducing defects, it is possible to improve maintainability, simplify maintenance processes, and lower associated costs. Hence, maintainability can be considered a fundamental attribute that directly affects software performance and quality. Predicting software maintainability involves identifying potential defects and errors that may occur during software operation. Such predictions enable developers to proactively avoid possible issues thereby improving software design and structure. This capability is particularly important as software maintainability continues to be one of the most significant challenges in software engineering. In recent years, artificial intelligence and machine learning techniques have provided advanced methods for defect prediction, supporting improved software quality through more effective and extensive testing strategies.

2. Related Work

2.1 Hybrid and Swarm Intelligence-Based MPPT Techniques

Recent research in Maximum Power Point Tracking (MPPT) has extensively explored hybrid and swarm intelligence algorithms to enhance tracking efficiency and convergence speed. Liao et al. [1] introduced a hybrid approach combining the bat algorithm (BA) with the discard strategy of the cuckoo search (CS) algorithm, achieving superior efficiency and tracking performance compared to conventional BA and other MPPT methods.

Chao et al. [2] proposed a dual-population strategy integrating genetic algorithm (GA) and ant colony optimization (ACO), where GA improved global exploration while ACO enhanced local search capability. This hybrid model demonstrated improved speed and efficiency over traditional approaches such as Perturb and Observe (P&O) and standard ACO.

Chtita et al. [3] further advanced hybrid optimization by combining Grey Wolf Optimizer (GWO) with Particle Swarm Optimization (PSO), achieving higher tracking accuracy, faster convergence to the global maximum power point (GMPP), and improved overall efficiency.

2.2 Improved Metaheuristic Optimization Algorithms

Several studies have focused on enhancing existing optimization algorithms to improve adaptability and performance under dynamic conditions. Refaat et al. [4] developed a self-tuning PSO (ST-PSO) algorithm that dynamically adjusts cognitive and social parameters, incorporating an inverse tangent function to constrain the search space and improve stability.

Gundogdu et al. [5] proposed an improved GWO algorithm with a dynamic movement adjustment mechanism based on the distance to the target, enabling faster convergence compared to the conventional fixed-rate approach.

Chao et al. [6] refined the cat swarm optimization (CSO) algorithm by introducing a slope-based adaptive step size mechanism, which improves tracking performance under varying irradiance and partial shading conditions. Similarly, Jabbar et al. [7] enhanced the water cycle optimization (WCO) algorithm using dynamic control parameters and chaotic mapping, resulting in faster convergence and reduced power losses.

Teshome et al. [8] proposed a modified firefly algorithm (FA) that calculates a representative position instead of following multiple brighter individuals, thereby reducing computational complexity and improving convergence speed.

2.3 Emerging Bio-Inspired Optimization Techniques

Recent advancements have also focused on improving newer bio-inspired algorithms for MPPT applications. Mohammed et al. [9] enhanced the snake optimizer (SO) by incorporating a fast-response mechanism to distinguish between uniform shading conditions (USCs) and partial shading conditions (PSCs), reducing redundant computations and improving tracking speed.

Mo et al. [10] refined the mayfly algorithm (MF) by introducing a gravity coefficient similar to the inertia weight in PSO, which improved convergence speed, stability, and tracking precision while minimizing oscillations.

Millah et al. [11] further improved GWO by integrating a pouncing strategy, significantly enhancing MPPT accuracy and response time.

2.4 Machine Learning and Ensemble-Based Approaches

In addition to optimization-based techniques, machine learning (ML) and deep learning (DL) approaches have been explored for MPPT applications. Tong et al. [12] highlighted that despite significant progress, DL models require extensive hyperparameter tuning, and improper configuration can lead to suboptimal performance.

To overcome these limitations, Borandag et al. [13] proposed a feature-selection-based ensemble approach that combines predictions using majority voting derived from information gain, uncertainty, and ReliefF features. Their results demonstrated that ensemble models consistently outperform traditional single-model ML approaches.

3. PROPOSED METHODOLOGY

The system architecture processes software metric data through a structured pipeline to perform predictive analysis of bug prediction and software type classification. It begins with dataset input and preprocessing, followed by Exploratory Data Analysis (EDA) to understand data patterns and relationships. The system incorporates baseline models such as BNB, MNB, and LDA, along with a proposed hybrid model HE-WSO-CNN-ET to enhance prediction accuracy as shown in Fig. 2. Model evaluation is carried out using standard performance metrics before deployment. The trained models are integrated into a Flask-based web application for real-time usage. The system also supports both single and batch predictions, making it practical for real-world scenarios. Additionally, an SQLite database is used for user authentication and data handling.

AMERICAN JOURNAL OF AI CYBER COMPUTING MANAGEMENT
Peer Reviewed, Rferred & Indexed Journal
E-ISSN:3069-0102
VOL.6, NO. 2(2026)

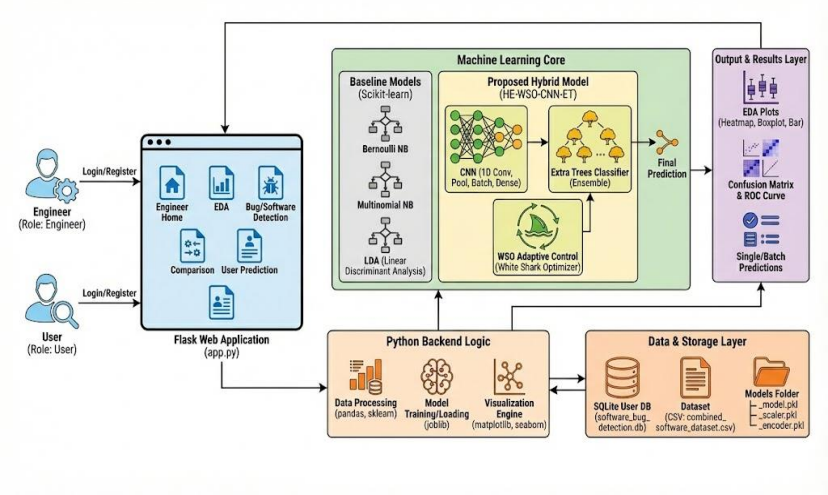


Fig. 2: System architecture of fine-grained software bug detection.

1. Data Collection and Input: The process starts with collecting software metric datasets in CSV format, which include various code and defect-related features. These datasets serve as the foundation for the system. Proper data organization ensures smooth execution of subsequent steps. The quality of input data significantly affects prediction outcomes.

2. Data Preprocessing and Feature Engineering: In this stage, the dataset is cleaned, and important features are selected for analysis. Data transformation techniques such as encoding and scaling are applied to standardize the inputs. This improves the learning capability of models. Proper preprocessing helps in reducing noise and increasing model efficiency.

3. EDA: EDA is performed using visualization techniques like heatmaps, boxplots, and distribution graphs. This helps in identifying patterns, correlations, and outliers in the dataset. The insights obtained assist in better feature understanding. It also supports informed model selection decisions.

4. Model Training and Selection: The processed data is split into training and testing sets for building predictive models. Baseline models such as BNB, MNB, and LDA are trained alongside the hybrid HE-WSO-CNN-ET model. The hybrid model improves prediction performance by combining multiple approaches. The best model is chosen based on evaluation results.

5. Model Evaluation and Storage: Models are evaluated using metrics such as accuracy, precision, recall, F1-score, and ROC curves. These measures ensure that the models perform reliably. After evaluation, the trained models are saved using joblib. This enables quick reuse without retraining.

6. Deployment and Prediction Interface: The system is deployed using a Flask web framework that provides an interactive interface for users. It allows both single and batch predictions based on user input. The application loads saved models and processes incoming data to generate predictions. An SQLite database is used to manage user authentication and system data efficiently.

HE-WSO-CNN-ET Classifier

HE-WSO-CNN-ET is a hybrid ensemble learning framework designed to combine deep feature learning, bio-inspired optimization, and ensemble decision-making into a unified prediction system. The model processes normalized software metrics to capture complex relationships that traditional classifiers may fail to identify. By integrating convolutional learning with an ensemble tree classifier, the framework achieves both expressive feature representation and robust classification performance. The adaptive optimization mechanism further stabilizes learning behaviour and enhances

generalization. This integrated workflow as illustrated in Fig. 3 allows the model to deliver accurate and reliable predictions for software-related classification tasks.

Input Feature Reception: The model receives pre-processed and normalized software metric features as input. These features represent complexity, coupling, cohesion, inheritance, and size characteristics. The input is structured to ensure compatibility with deep learning and ensemble components.

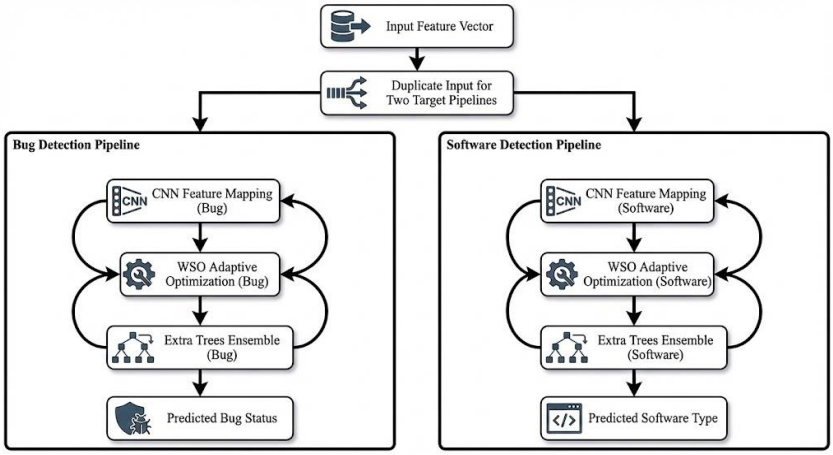


Fig. 3: Internal workflow of HE-WSO-CNN-ET.

CNN-Based Feature Mapping: The input features are reshaped and passed through a one-dimensional CNN. Convolutional layers learn local feature interactions and hidden patterns among software metrics. This process generates a transformed feature representation that captures higher-level abstractions.

Feature Enhancement and Flattening: The extracted convolutional features are normalized and flattened into a compact feature vector. This step converts deep representations into a form suitable for ensemble learning. It ensures that learned patterns are preserved while maintaining computational efficiency.

WSO–Inspired Adaptive Control: An adaptive control mechanism inspired by the WSO is applied to guide learning stability. This component simulates population-based adaptive behaviour to regulate internal parameter dynamics. It enhances convergence behaviour and reduces sensitivity to noise in feature representations.

ET Ensemble Classification: The enhanced feature vectors are forwarded to an ET ensemble classifier. Multiple randomized decision trees independently learn decision boundaries based on feature variations. Their collective output produces a robust and generalized prediction.

Final Prediction Generation: The ensemble outputs are aggregated to generate the final class prediction. The model assigns the input to the class with the highest confidence score. This decision represents the combined intelligence of deep learning, optimization, and ensemble reasoning.

4. Results description

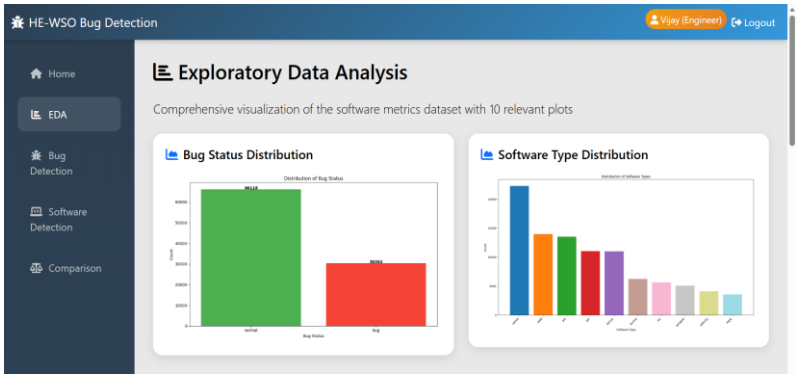


Fig. 4: EDA Interface of the HE-WSO-Based Software Bug Detection System.

Fig. 4 illustrates the exploratory data analysis interface used to examine the underlying characteristics of the software metrics dataset. This interface presents multiple analytical visualizations that help in understanding bug distribution and software type frequency across the dataset. It supports statistical exploration by revealing patterns, trends, and imbalances present in the data prior to model training. The visual summaries assist engineers in gaining insights into data behaviour and feature relevance.

Fig. 5: Feature correlation and feature mean analysis in the software metrics dataset

Fig. 5 illustrates the analytical visualizations used to examine relationships and statistical behaviour among software metrics. The correlation heatmap highlights the degree of association between different features, helping to identify strongly related and weakly related metrics. The feature mean analysis compares average metric values across different bug status categories, revealing how software attributes vary between normal and defective instances. These visualizations support deeper understanding of feature dependencies and discriminative characteristics within the dataset.

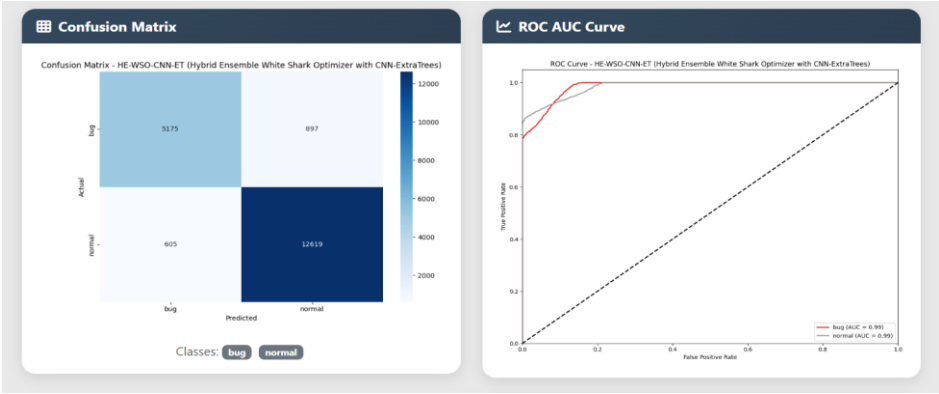


Fig. 6: Confusion Matrix and ROC–AUC Curve for HE-WSO-CNN-ET in Bug Detection

Fig. 6 illustrates the performance evaluation of the proposed HE-WSO-CNN-ET hybrid ensemble model for the software bug detection task. The confusion matrix demonstrates a high number of correctly classified buggy and normal instances, indicating strong predictive accuracy and reduced misclassification. The ROC–AUC curve shows a near-ideal performance with a significantly high area under the curve, reflecting excellent discriminative capability between defective and non-defective software components. These results highlight the effectiveness of integrating deep feature learning, bio-inspired optimization, and ensemble classification.

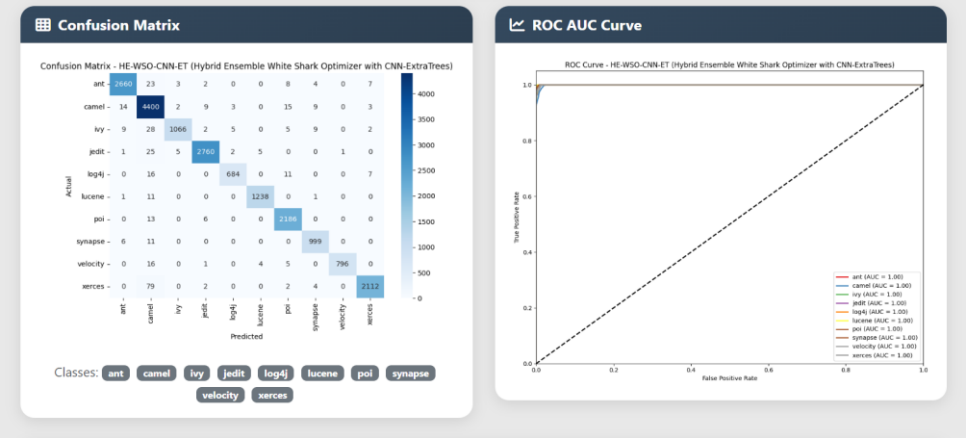


Fig. 7: Confusion Matrix and ROC–AUC Curve for HE-WSO-CNN-ET in Software Type Classification

Fig. 7 illustrates the performance evaluation of the proposed HE-WSO-CNN-ET model for multi-class software type classification. The confusion matrix shows a strong concentration of predictions along the diagonal, indicating a high level of correct classification across all software categories. The ROC–AUC curves demonstrate near-perfect discriminative performance, with area-under-curve values approaching unity for all classes. These results highlight the model's ability to effectively capture complex patterns and class distinctions within the software metrics dataset.

Comparative Analysis

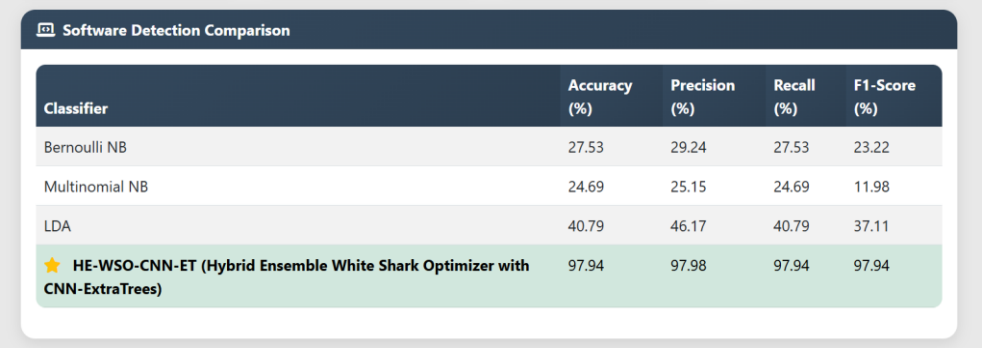
The comparative analysis section presents a systematic evaluation of different machine learning models used for software bug detection and software type classification. This analysis aims to highlight variations in predictive performance, robustness, and discriminative capability across baseline and hybrid models. Traditional classifiers are compared with the proposed hybrid ensemble approach using standard evaluation metrics and visual performance indicators. The comparison provides insights into how different learning paradigms handle complex software metric data. By examining both binary and multi-class prediction tasks, this analysis helps identify the strengths and limitations of each model.

Bug Detection Comparison

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Bernoulli NB	65.37	63.8	65.37	64.4
Multinomial NB	69.08	71.94	69.08	57.34
LDA	69.96	67.24	69.96	62.96
⭐ **HE-WSO-CNN-ET (Hybrid Ensemble White Shark Optimizer with CNN-ExtraTrees)**	92.22	92.16	92.22	92.16

Fig. 8: Comparative performance analysis of models for bug detection

Fig. 8 illustrates the comparative performance of different machine learning models evaluated for the software bug detection task. The comparison is carried out using standard evaluation metrics including accuracy, precision, recall, and F1-score BNB achieves an accuracy of 65.37%, while MNB and LDA record accuracies of 69.08% and 69.96%, respectively, indicating moderate predictive capability. In contrast, the proposed HE-WSO-CNN-ET model attains a significantly higher accuracy of 92.22%, along with consistently superior precision, recall, and F1-score values. This substantial improvement demonstrates the effectiveness of the hybrid ensemble approach in accurately identifying buggy software modules compared to traditional baseline models.



Software Detection Comparison

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Bernoulli NB	27.53	29.24	27.53	23.22
Multinomial NB	24.69	25.15	24.69	11.98
LDA	40.79	46.17	40.79	37.11
⭐ HE-WSO-CNN-ET (Hybrid Ensemble White Shark Optimizer with CNN-ExtraTrees)	97.94	97.98	97.94	97.94

Fig. 9: comparative performance analysis of models for software type detection

Fig. 9 illustrates the comparative performance evaluation of different machine learning models applied to the software type detection task. The analysis reports standard evaluation metrics including accuracy, precision, recall, and F1-score for each classifier. BNB and MNB achieve low accuracies of 27.53% and 24.69%, indicating limited effectiveness in handling multi-class software classification. LDA improves performance with an accuracy of 40.79%, demonstrating better class separation. In contrast, the proposed HE-WSO-CNN-ET model achieves a substantially higher accuracy of 97.94%, along with consistently superior precision, recall, and F1-score values. This comparison clearly highlights the dominance of the hybrid ensemble approach in accurately classifying software types.

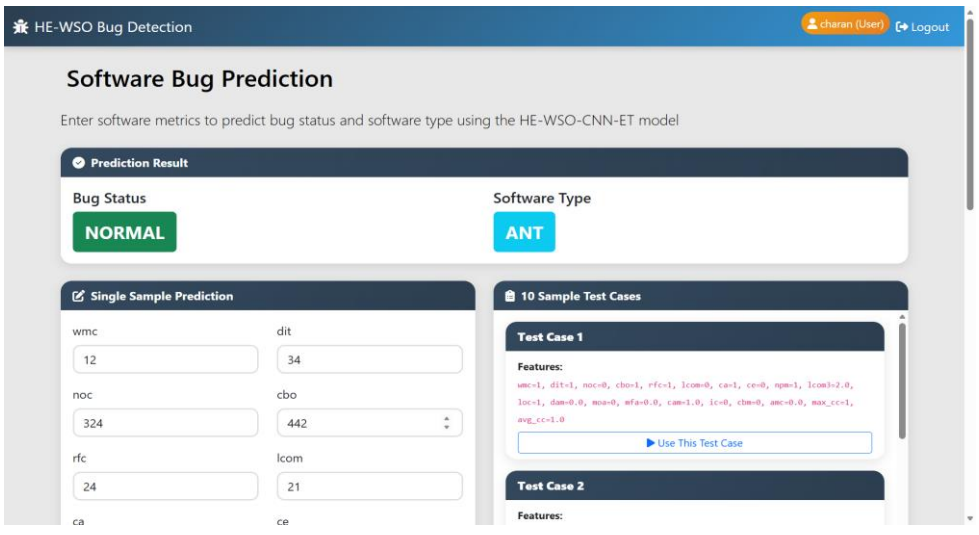


Fig. 10: Prediction result interface for software bug and software type detection

Fig. 10 illustrates the prediction result interface that displays the output generated by the HE-WSO-CNN-ET model based on user-provided software metrics. This interface presents the predicted bug status and the corresponding software type for the given input instance. It reflects the model's capability

to perform simultaneous bug detection and software classification within a unified framework. The displayed results enable users to directly interpret the outcome of the prediction process.

Fig. 11 illustrates the batch prediction interface that enables users to perform large-scale prediction by uploading a CSV file containing multiple software instances. This interface supports efficient processing of bulk software metrics to simultaneously predict bug status and software type using the trained HE-WSO-CNN-ET model. It facilitates practical deployment by allowing users to analyse multiple records in a single operation rather than individual inputs. The batch prediction mechanism enhances scalability and usability of the system for real-world software datasets.

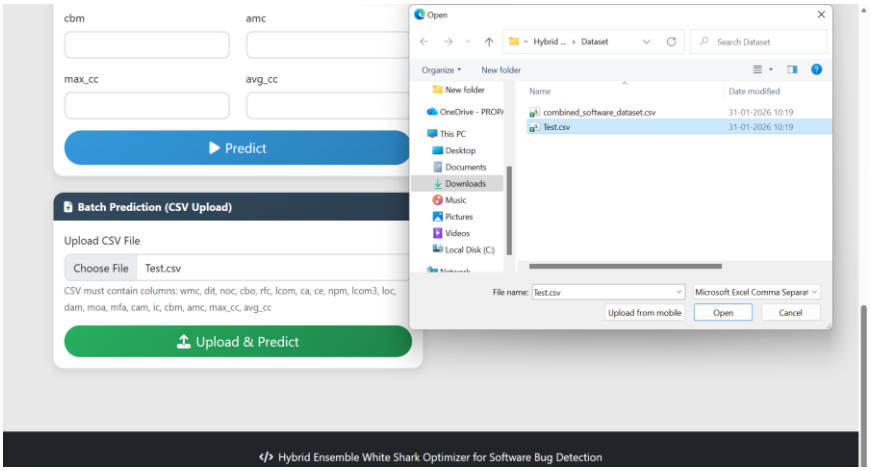


Fig. 11: Batch prediction interface for software bug and software type detection

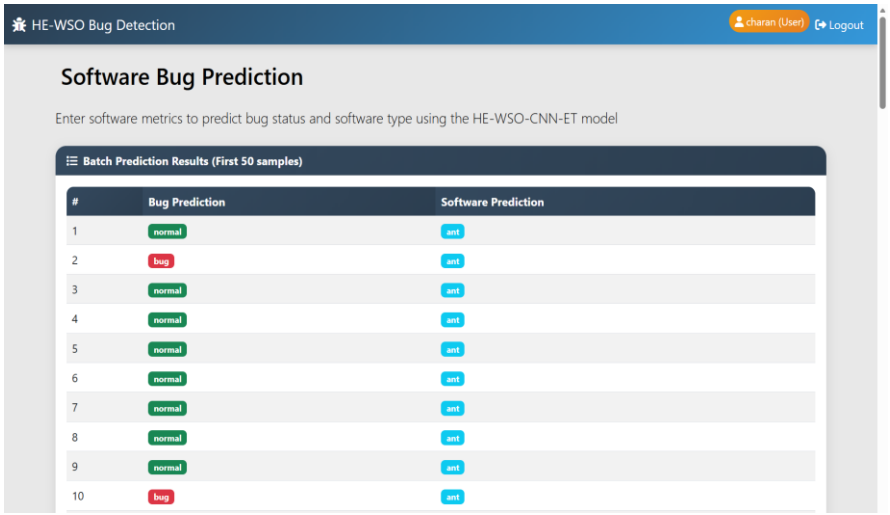


Fig. 12: Batch prediction results for software bug and software type detection

Fig. 12 illustrates the batch prediction results generated by the HE-WSO-CNN-ET model for multiple software instances processed simultaneously. The results present predicted bug status alongside the corresponding software type for each input record. This output demonstrates the model's capability to handle large-scale prediction efficiently while maintaining consistent decision accuracy. It enables users to quickly analyse trends and outcomes across multiple software components.

5. Conclusion

This study presented an intelligent software bug detection and software type classification system built using a HE-WSO-CNN-ET. The integration of deep feature learning through CNN, adaptive

optimization via WSO, and robust ensemble decision-making using Extra Trees significantly enhanced predictive performance compared to traditional machine learning models. Experimental results demonstrate substantial improvements in accuracy, precision, recall, and F1-score for both bug detection and software detection tasks, with the proposed model achieving superior generalization and discriminative capability. The comparative analysis clearly shows that HE-WSO-CNN-ET effectively captures complex nonlinear relationships among software metrics, reduces misclassification, and provides reliable predictions. The system's web-based implementation further validates its practical applicability for real-world software quality analysis and decision support.

REFERENCES

[1] Liao, C.Y.; Subroto, R.K.; Millah, I.S.; Lian, K.L.; Huang, W.T. An Improved Bat Algorithm for More Efficient and Faster Maximum Power Point Tracking for a Photovoltaic System Under Partial Shading Conditions. *IEEE Access* **2020**, *8*, 96378–96390.

[2] Chao, K.H.; Rizal, M.N. A Hybrid MPPT Controller Based on the Genetic Algorithm and Ant Colony Optimization for Photovoltaic Systems under Partially Shaded Conditions. *Energies* **2021**, *14*, 2902.

[3] Chtita, S.; Motahhir, S.; El Hammoumi, A.; Chouder, A.; Benyoucef, A.S.; El Ghzizal, A.; Derouich, A.; Abouhawwash, M.; Askar, S. A novel hybrid GWO–PSO-based maximum power point tracking for photovoltaic systems operating under partial shading conditions. *Sci. Rep.* **2022**, *12*, 10637.

[4] Refaat, A.; Elbaz, A.; Khalifa, A.E.; Mohamed Elsakka, M.; Kalas, A.; Hegazy Elfar, M. Performance evaluation of a novel self-tuning particle swarm optimization algorithm-based maximum power point tracker for porton exchange membrane fuel cells under different operating conditions. Energy Convers. Manag. 2024, 301, 118014.

[5] Gundogdu, H.; Demirci, A.; Tercan, S.M.; Cali, U. A Novel Improved Grey Wolf Algorithm Based Global Maximum Power Point Tracker Method Considering Partial Shading. IEEE Access 2024, 12, 6148–6159.

[6] Chao, K.H.; Nguyen, T.B.N. Global Maximum Power Point Tracking of a Photovoltaic Module Array Based on Modified Cat Swarm Optimization. Appl. Sci. 2024, 14, 2853.

[7] Jabbar, R.I.; Mekhilef, S.; Mubin, M.; Alshammari, O.; Kazaili, A. A Fast MPPT Method Based on Improved Water Cycle Optimization Algorithm for Photovoltaic Systems Under Partial Shading Conditions and Load Variations. IEEE Open J. Ind. Electron. Soc. 2024, 5, 1324–1338.

[8] Teshome, D.F.; Lee, C.H.; Lin, Y.W.; Lian, K.L. A Modified Firefly Algorithm for Photovoltaic Maximum Power Point Tracking Control Under Partial Shading. IEEE J. Emerg. Sel. Top. Power Electron. 2017, 5, 661–671.

[9] Mohammed, K.K.; Mekhilef, S. Improved Snake Optimizer Algorithm-Based GMPPT With a Fast Response to the Load Variations Under Different Weather Conditions for PV Systems. IEEE Trans. Ind. Electron. 2024, 71, 7147–7157.

[10] Mo, S.; Ye, Q.; Jiang, K.; Mo, X.; Shen, G. An improved MPPT method for photovoltaic systems based on mayfly optimization algorithm. Energy Rep. 2022, 8, 141–150.

[11] Millah, I.S.; Chang, P.C.; Teshome, D.F.; Subroto, R.K.; Lian, K.L.; Lin, J.F. An Enhanced Grey Wolf Optimization Algorithm for Photovoltaic Maximum Power Point Tracking Control Under Partial Shading Conditions. *IEEE Open J. Ind. Electron. Soc.* 2022, *3*, 392–408.

[12] Mohd Adnan, M.; Sarkheyli, A.; Mohd Zain, A.; Haron, H. Fuzzy logic for modeling machining process: A review. *Artif. Intell*, *43*, 345–379.

[13] Elish, M.O.; Aljamaan, H.; Ahmad, I. Three empirical studies on predicting software maintainability using ensemble methods. *Soft Comput*, *19*, 2511–2524.